HW2 Report

Roșu Cristian-Mihai

December 2, 2019

Abstract

Genetic algorithms have a lot of properties that make them a good choice when one needs to solve very complicated problems, however heuristics are powerful alternatives. One such problem is finding the global minima/maxima of multidimensional functions.

This work explores the efficiency of a genetic algorithm when dealing with exactly this problem and comparing it to two of the most representative heuristic search methods: Iterated Hillclimbing and Simulated Annealing.

1 Introduction

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. [1]

This work aims to explore the capabilities of such a genetic algorithm to generate solutions for the problem of searching for the global minimum of a multidimensional function.

Here, *Rastrigin*'s, *Rosenbrock*'s, *De Jong*'s and *Michalewicz*'s functions are used as benchmarks to test a simple genetic algorithm. Below are the functions [2], in order:

$$f(x) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot cos(2\pi x_i)\right], A = 10, x_i \in [-5.12, 5.15]$$

$$f(x) = \sum_{i=1}^{n-1} \left[100({x_i}^2 - x_{i+1})^2 + (x_i - 1)^2\right], x_i \in [-5, 10]$$

$$f(x) = \sum_{i=1}^{n} {x_i}^2, x_i \in [-5.12, 5.12]$$

$$f(x) = -\sum_{i=1}^{n} \left[\sin(x_i^2) \cdot \sin\left(\frac{i{x_i}^2}{\pi}\right)^{2m}\right], m = 10, x_i \in [0, \pi]$$

Being multidimensional, each of the functions will be tested on 5, 10 and 30 dimensions, and for every test the *minimum*, *maximum* and *average* **values** found, as well as the *minimum*, *maximum* and *average* **time** it took to obtain those results will be recorded.

1.1 Motivation

The purpose of this work is to explore the capabilities of a genetic algorithm.

In that sense, the problem of finding a function's global minimum has been chosen due to its ease of implementation and of understanding on a theoretical level. And as for the algorithms chosen, Iterated Hillclimbing and Simulated Annealing are the cornerstones of what genetics represents in computer science, on top of being equally easy to understand and implement. As such, they are the perfect candidates to be compared to a genetic algorithm.

The benchmark functions have been chosen such that the experiment may include as much variety as possible, since the domain and global minimum of each of them are very distinct.

2 Method

In order to find the global minimum of a function, we use a simple genetic algorithm and incorporate said function into the algorithm.

This is the general structure of a genetic algorithm:

```
population <- vector(popSize, randomVector(d * L));
while (! STOP_CONDITION) {
    mutation(population);
    crossover(population);
    population <- selection(population, f());
}
```

By trying to imitate nature, a genetic algorithm considers the representation of a population and emulates evolution across a certain number of generations from which to extract solutions to the problem at hand.

In this case, the population is represented in a bitstring that is initially randomly generated called `population` whose size depends on the number of individuals in the population, or *cromozomes*, and the length of their representation, a bitstring as well. The number of cromozomes used is 100.

After that, in order to manufacture the solution, the genetic operators are applied to the population:

- **Mutation** - each bit in `population` has the low chance of **0.01** of being inverted in order to increase diversity in the population and eliminate the posibility of stagnation.
- **Crossover** - the operator with the biggest impact since it adds new cromozomes in the population. Each cromozome is given a random chance,

from 0 to 1. They are then sorted according to this chance and paired. The pairs whose chances are lower than **0.3**, the chance for crossover, participate in crossover and create 2 new cromozomes by interchanging bits after a randomly chosen *cutpoint*.

- **Selection** - the operator that chooses the cromozomes that will be part of the next generation. In order to determine that, a fitness is calculated for each of the cromozomes. That fitness is where the optimization function comes into play since the cromozome whose function evaluation is of lower value has a better fitness. That fitness is calculated with the following formula:

  `fitness(`x_i`) <- 1.1 * max - f(`x_i`),` $\forall x_i$ `cromozome`

  where `max` is the maximum function value out of all the cromozomes in the current generation and `f()` is the optimization function.

  The process of choosing a candidate to go into the next generation is done using the **Roulette Wheel** method coupled with **elitism**: the first **5** cromozomes with the best fitness are carried over and the rest are chosen randomly using a probabilty field that favours the cromozomes with high fitness.

These operations are performed on the population until a certain condition is met. In this case, the stop condition is when the population reaches its **250th generation**. The best cromozome out of all the generations is the *global minimum* found by the algorithm.

3 Experiment

The experiment consists in running the genetic algorithm through each of the benchmark functions on **5**, **10** and **30** dimensions, over **15 iterations**, in order to get a sample big enough to compare them by looking at the minimum, maximum and average *values* produced, and the minimum, maximum and average *times* it took to reach those values. In order to slightly speed up this process, the tests are run on two threads.

The performance of the genetic algorithm is then graphically compared to the results of the previous experiment in HW1.

The code for this can be found on Github [3].

4 Results

Below are 4 tables corresponding to each of the 4 functions that hold the results to the experiment: Rastrigin, Rosenbrock, De Jong and Michalewicz, in that order.

Method	Dimension n	5	10	30
Genetic Algorithm	Min	0	4.5997	73.8115
	Max	1.2481	11.4434	133.3624
	Average	0.6240	6.3535	99.7060
	Min time	212s	413s	1233s
	Max time	302s	580s	1769s
	Average time	288s	560s	1718s

Figure 1: Ratrigin's function results

Method	Dimension n	5	10	30
Genetic Algorithm	Min	0.0347	17.0955	4732.8138
	Max	5.1822	193.9965	40878.7201
	Average	2.0764	74.9848	14827.7693
	Min time	200s	431s	1235s
	Max time	294s	607s	1819s
	Average time	283s	584s	1720s

Figure 2: Rosenbrock's function results

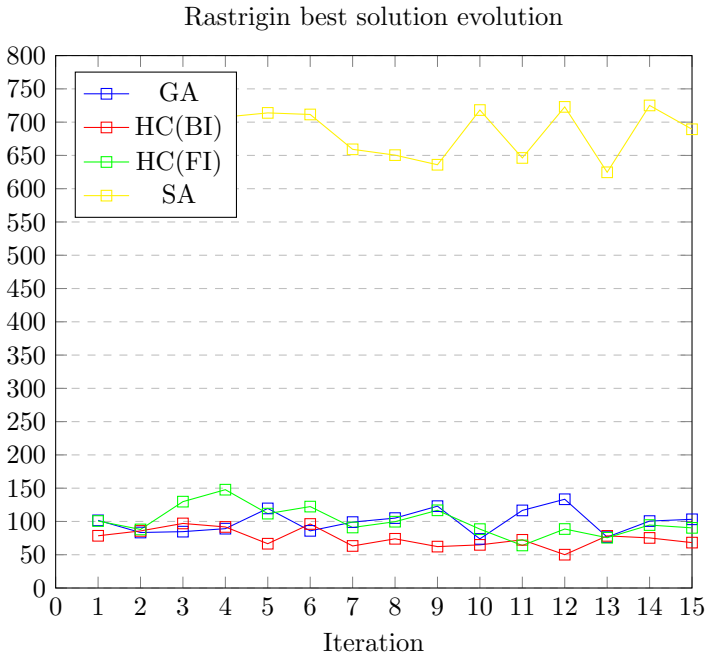
Method	Dimension n	5	10	30
Genetic Algorithm	Min	0	0.0028	3.0435
	Max	0	0.0163	5.5702
	Average	0	0.0087	4.1344
	Min time	244s	411s	1285s
	Max time	335s	599s	3127s
	Average time	315s	580s	1898s

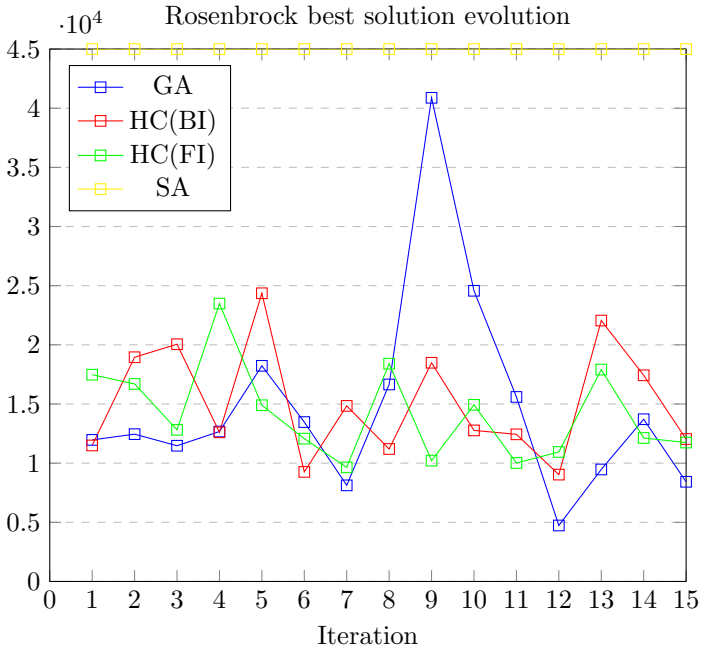
Figure 3: De Jong's function results

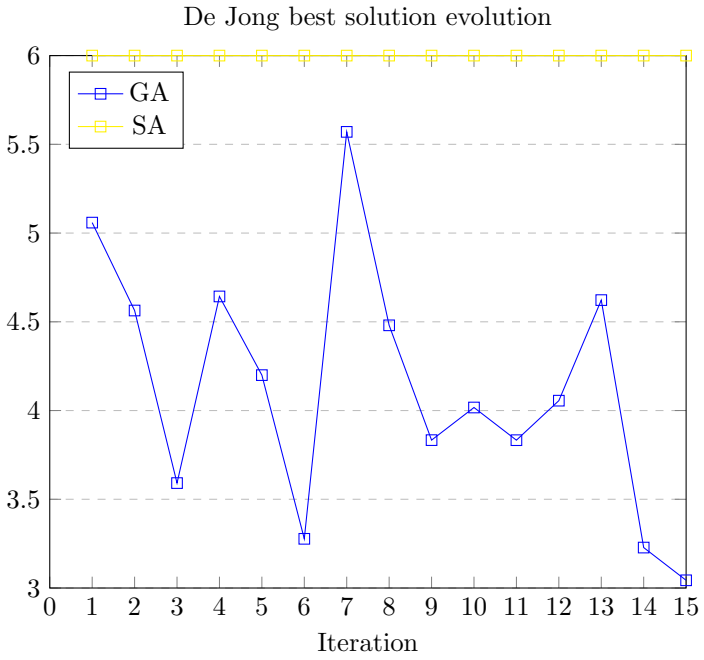
Method	Dimension n	5	10	30
Genetic Algorithm	Min	-4.6879	-9.4187	-20.5711
	Max	0	0	0
	Average	-4.5961	-9.1553	-19.4458
	Min time	212s	419s	1213s
	Max time	291s	575s	1716s
	Average time	284s	562s	1637s

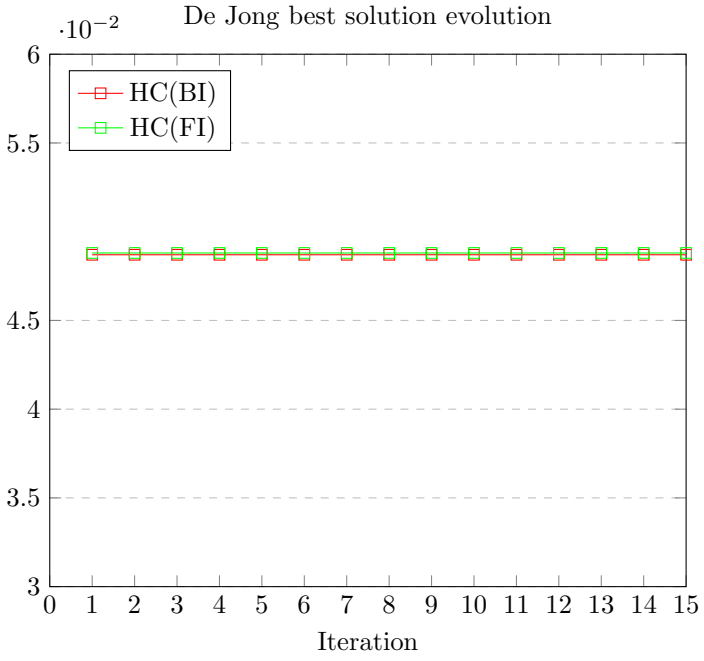
Figure 4: Michalewicz's function results

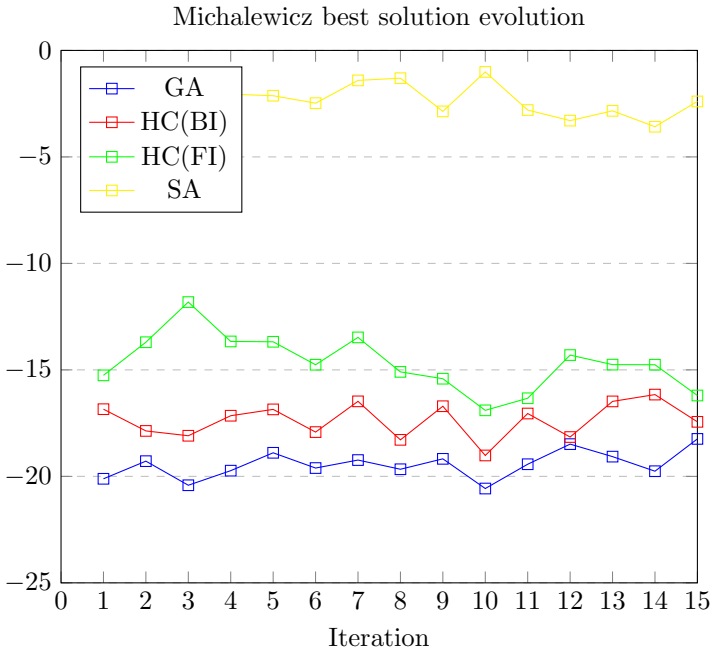
And below are the graphs showing the difference in performance on 30 dimensions between the genetic algorithm and the heuristic search methods Iterated Hillclimbing and Simulated Annealing:


Rastrigin best solution evolution
GA
HC(BI)
HC(FI)
SA
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Iteration


Rosenbrock best solution evolution
·10^4
GA
HC(BI)
HC(FI)
SA
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Iteration


De Jong best solution evolution
GA
SA
6
5.5
5
4.5
4
3.5
3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Iteration


De Jong best solution evolution
$\cdot 10^{-2}$
HC(BI)
HC(FI)
6
5.5
4.5
4
3.5
3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Iteration


Michalewicz best solution evolution
GA
HC(BI)
HC(FI)
SA
0
−5
−10
−15
−20
−25
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Iteration

5 Conclusions

The genetic algorithm provides steadier and more consistent results, as oposed to the two heuristic search algorithms. Across the 4 different function no parameter changes have been made and still the solutions found and the time needed to reach them are fairly proportional. Additionaly, the genetic algorithm has produced very good results on the lower dimensions.

On the other hand, it looses precision rather quickly as the dimension increases, and the times become a great deal longer, on top of already being so compared to the alternatives.

References

[1] Wikipedia page for Genetic algorithm
`https://en.wikipedia.org/wiki/Genetic_algorithm`

[2] Site with details of the functions used
`http://www.geatbx.com/docu/fcnindex-01.html#P150_6749`

[3] Github repository for the project
`https://github.com/Nenma/ga-hw2`

[4] Course site with the algorithms' details
`https://profs.info.uaic.ro/~pmihaela/GA/laborator2.html`

[5] Simple Latex tutorial used
`http://www.docs.is.ed.ac.uk/skills/documents/3722/3722-2014.pdf`

[6] Thesis on GA Optimization used as reference
`https://www.diva-portal.org/smash/get/diva2:832349/FULLTEXT01.pdf`